Integrating the Par Lab Stack
Running Damascus on SEJITS/ROS/RAMP Gold

Kevin Klues, Yunsup Lee, Andrew Waterman

Par Lab Winter Retreat 2010
Integrating the Stack

• Overall Goal of the Par Lab:
  Create Productive, Efficient, Correct, Portable SW for 100+ cores that scales as the core count increase every 2 years (!)

• Tremendous Progress has been made at individual layers in the Par Lab Software Stack

• This Demo is an attempt to demonstrate our integration effort across multiple layers
Integrating the Stack

- Personal Health
- Image Retrieval
- Hearing, Music
- Speech
- Parallel Browser

Design Patterns/Motifs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

Parallel Libraries

Parallel Frameworks

Efficiency Languages

Sketching

Autotuners

Legacy Code

Schedulers

Communication & Synch. Primitives

Efficiency Language Compilers

Legacy OS

OS Libraries & Services

Hyervisor

Multicore/GPGPU

ParLab Manycore/RAMP

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Correctness

Productivity Layer

Diagnosing Power/Performance

Applications

Efficiency Layer

OS

Arch.
Integrating the Stack

Damascene

- Parallel implementation of the Global Probability of Boundary (gPB) image contour detection algorithm
Integrating the Stack

Damascene
- Parallel implementation of the Global Probability of Boundary (gPB) image contour detection algorithm

Nelson Morgan
Integrating the Stack

Damascene

- Parallel implementation of the Global Probability of Boundary (gPB) image contour detection algorithm
Integrating the Stack

Damascene

- Parallel implementation of the Global Probability of Boundary (gPB) image contour detection algorithm

Nelson Morgan

TG, R=5

Full gPB
Integrating the Stack

Damascene

- Parallel implementation of the Global Probability of Boundary (gPB) image contour detection algorithm
- Algorithm developed by Prof. Jitendra Malik’s Vision Group at UCB
- Bryan Catanzaro et al. parallelized the code using CUDA, including some algorithmic changes (presented at last year’s winter retreat)
- We’ve now parallelized the same code in C and gotten it running in the python/SEJITS framework
- This demo shows this parallelization effort in action
- More details to come.....
We picked a small subset of the full Damascus code and implemented it for SEJITS in python using two parallel constructs.
Integrating the Stack

Python/SEJITS

- We picked a small subset of the full Damascene code and implemented it for SEJITS in python using two parallel constructs.
- Since SEJITS targets x86, NVIDIA GPUs, and SPARC v8 (i.e. RAMP Gold) the Damascene application can now run on all three platforms.
Integrating the Stack

RAMP Gold
- Par Lab architecture simulator on an FPGA
- 64-core SPARC V8 target machine
- Runtime-configurable memory hierarchy
- 50 MIPS on Xilinx LX110T
  - 269x faster than comparable software simulator (SIMICS+GEMS)
- Sufficient HW support (MMU, timer, ...) to boot ROS, Linux 2.6
Integrating the Stack

ROS (Tessellation)
- New operating system designed for improved kernel scalability and better support for parallel applications
Integrating the Stack

ROS (Tessellation)
- New operating system designed for improved kernel scalability and better support for parallel applications
- Built around idea of Space-Time Partitioning of Resources
  - Resources partitioned amongst execution entities based on explicit requests
  - Execution entities scheduled in both space and time based on meeting resource guarantees
  - Enables predictable application performance
  - Increases isolation
Integrating the Stack

ROS (Tessellation)
- New operating system designed for improved kernel scalability and better support for parallel applications
- Built around idea of Space-Time Partitioning of Resources
  - Resources partitioned amongst execution entities based on explicit requests
  - Execution entities scheduled in both space and time based on meeting resource guarantees
  - Enables predictable application performance
  - Increases isolation
- The Cell/Partition execution model
  - Multiple cores ‘owned’ by a single cell
  - All cores gang scheduled
  - Exposed via the bthreads/harts ABI
Integrating the Stack

C Runtime / Lithe
- Support for **newlib**, **glibc**, and **parlib**
- **parlib** is our internally developed C runtime supporting our new OS abstractions
Integrating the Stack

C Runtime / Lithe
- Support for **newlib**, **glibc**, and **parlib**
  - **parlib** is our internally developed C runtime supporting our new OS abstractions
- Lithe support
  - Liquid threads framework
  - Enables multiple application thread schedulers to run cooperatively
  - Not yet fully integrated (but soon!)
- **bthreads** / harts ABI
  - Tightly integrated with the OS, providing a userspace abstraction on top of its cell/partition model
Integrating the Stack

**C Runtime / Lithe**
- Support for **newlib**, **glibc**, and **parlib**
  - **parlib** is our internally developed C runtime supporting our new OS abstractions
- Lithe support
  - Liquid threads framework
  - Enables multiple application thread schedulers to run cooperatively
  - Not yet fully integrated (but soon!)
- bthreads / harts ABI
  - Tightly integrated with the OS, providing a userspace abstraction on top of its cell/partition model
- Also support for legacy OS integration
DEMO
Damascene: Intermediate Steps

RGB -> L*a*b Color Conversion

Convolve Filters

K-means

filters

textons

Multiple steps

TG, R=5

Multiple steps
Paths Through the Stack

- Damascusne
- Python/SEJITS
- C Runtime
- Linux
- x86
Paths Through the Stack

- Damascus
- Native C
- C Runtime
- ROS
- RAMP Gold
Paths Through the Stack

- Damascusne
- Native C
- C Runtime
- ROS
- RAMP Gold

Time for the Demo!
(and questions while we set up)